Algorithm Selection of Anytime Algorithms*

Alexandre D. Jesus
Univ. Lille, CNRS, Centrale Lille, Inria,
UMR 9188 - CRIStAL, F-59000 Lille, France
University of Coimbra, CISUC, DEI, Coimbra, Portugal
ajesus@dei.uc.pt

Arnaud Liefooghe
JFLI - CNRS IRL 3527, University of Tokyo, Tokyo, Japan
arnaud.liefooghe@univ-lille.fr

Bilel Derbel
Univ. Lille, CNRS, Centrale Lille, Inria,
UMR 9189 - CRIStAL, F-59000 Lille, France
bilel.derbel@univ-lille.fr

Luís Paquete
University of Coimbra, CISUC, DEI, Coimbra, Portugal
paquete@dei.uc.pt

ABSTRACT

Anytime algorithms for optimization problems are of particular interest since they allow to trade off execution time with result quality. However, the selection of the *best* anytime algorithm for a given problem instance has been focused on a particular budget for execution time or particular target result quality. Moreover, it is often assumed that these anytime preferences are known when developing or training the algorithm selection methodology. In this work, we study the algorithm selection problem in a context where the decision maker's anytime preferences are defined by a general utility function, and only known at the time of selection. To this end, we first examine how to measure the performance of an anytime algorithm with respect to this utility function. Then, we discuss approaches for the development of selection methodologies that receive a utility function as an argument at the time of selection. Then, to illustrate one of the discussed approaches, we present a preliminary study on the selection between an exact and a heuristic algorithm for a bi-objective knapsack problem. The results show that the proposed methodology has an accuracy greater than 96% in the selected scenarios, but we identify room for improvement.

CCS CONCEPTS

• **Computing methodologies** → Search methodologies; **Learning settings**;

KEYWORDS

Algorithm Selection · Anytime Algorithms · Anytime Performance Measure

*This is the author's version of the work. It is made available for your personal use. Not for redistribution. The definitive Version of Record was published in "GECCO '20: Proceedings of the 2020 Genetic and Evolutionary Computation Conference", https://doi.org/10.1145/3377930.3390185.

GECCO '20, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3390185

ACM Reference Format:
Alexandre D. Jesus, Arnaud Liefooghe, Bilel Derbel, and Luís Paquete. 2020. Algorithm Selection of Anytime Algorithms. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3390185

1 INTRODUCTION

Anytime algorithms [4, 21] are of interest in various domains since they allow a decision maker to trade-off execution time with solution quality. In addition, many anytime algorithms often exist for a given problem. For example, evolutionary algorithms and other search heuristics can often return a solution if interrupted at any time. Unfortunately, there is usually no single algorithm that always shows the best solution quality for all execution times and instances. As a result, when presented with a new problem instance, a decision maker is interested in selecting which algorithm to use.

When selecting the best anytime algorithm, a decision maker is typically also interested in defining its preferences with respect to the interruption of the algorithm, which we denote *anytime preferences*. For example, consider a real-time system that receives problem instances to solve. For some instances, the system needs to return a solution within a short amount of time, e.g. 1 second. For other instances, the system has more time, e.g. 60 seconds, while for others, the exact time depends on external factors that are not yet fully established, but it is known that the algorithm will be interrupted between two time points, e.g. between 1 and 60 seconds. Since the choice of an algorithm likely depends on the available time budget, a selection methodology should take this knowledge into account.

When the anytime preferences can change between calls to the selection methodology, as in the previous example, we say that we have *dynamic anytime preferences*. Otherwise, we say *static anytime preferences*. Anytime preferences can be defined with respect to measurements other than time, e.g. target solution quality, or with respect to multiple measurements, e.g. target solution quality and available time budget. For this work, we consider that the anytime preferences are characterized by a utility function $w : T \times Q \to \mathbb{R}_0^+$ that denotes how likely the algorithm is to be interrupted at a particular time $t \in T$ and solution quality $q \in Q$.

In this work, we are interested in studying the *algorithm selection problem* [18] for anytime algorithms, while allowing for dynamic anytime preferences. In particular, we study the following two aspects: (i) the definition of performance measures for anytime

algorithms that take into account the anytime preferences denoted by a utility function w; and (ii) the design of selection methodologies that consider dynamic anytime preferences. We note that numerous selection methodologies can be found in the literature, see [10, 11] for a review. However, up to our knowledge, none consider the characterization of the anytime preferences of the decision maker by a utility function, nor the setting of anytime preferences at the time of selection.

The remainder of this paper is organized as follows. In Section 2, we introduce anytime algorithms and notions related to the anytime performance of such algorithms. In Section 3, we describe performance measures for anytime algorithms with respect to possible definitions of the utility function w, and discuss their theoretical properties. In Section 4, we discuss the possible approaches to the algorithm selection problem with respect to dynamic anytime preferences. In Section 5, we present a preliminary study on the selection between an exact and a heuristic algorithm for a bi-objective knapsack problem to illustrate and study a concrete selection methodology based on one of the approaches discussed in Section 4. Finally, we summarize the results of this work and discuss possible directions for future work in Section 6.

2 ANYTIME ALGORITHMS

Anytime algorithms [4, 21] offer a trade off between execution time and solution quality, since they can return a solution when stopped at any time. Many existing algorithms to solve optimization problems are anytime since they can return the best solution found when interrupted, e.g. the best solution found so far by evolutionary algorithms and other search heuristics. Throughout this work we assume that the domain for execution time is denoted by a totally ordered set T, such that for any two time steps $t, t' \in T$ the relation $t < t'$ means that the moment t happens earlier than the moment t'. Likewise, the domain of solution quality is denoted by a totally ordered set Q, where the relation $q < q'$ between any two values $q, q' \in Q$ denotes that q is worse than q' to the decision maker.

To study the performance of a single run of an anytime algorithm we define a *performance trace* that describes the trade-off between solution quality and execution time. A run here refers to a single execution of an algorithm on a particular instance. The set of all runs of an algorithm a is denoted by Ω_a.

Definition 2.1 (Performance Trace). The *performance trace* of a run $r \in \Omega_a$ of an algorithm a is defined by a function $K_{a,r} : T \to Q$ that maps execution time to the quality of the solution that would be returned if the algorithm was interrupted at that time.

If every run of an anytime algorithm a gives a monotonic performance trace, that is

$$t_1 < t_2 \implies K_{a,r}(t_1) \geq K_{a,r}(t_2) \quad \forall r \in \Omega_a \ \ \forall t_1, t_2 \in T$$

then algorithm a is said to have *monotonic behavior*. Otherwise, algorithm a is said to have *non-monotonic behavior*. Algorithms with non-monotonic behavior can typically be made to have monotonic behavior by keeping the best solution in memory. On the other hand, this may not be possible if the quality of a solution cannot be easily measured. For example, consider an evolutionary algorithm to optimize a function. If this function is expensive to evaluate or if it is not available during the execution, the algorithm may need to consider an approximation function to evaluate the fitness of the individuals. If this approximation is not consistent with the original function, e.g. if a change to the decision variables results in an increase in quality for the approximation function but a decrease in quality for the real function, then the algorithm might replace solutions of higher real quality by solutions with worse real quality. As a result the algorithm may present non-monotonic behavior. The remaining definitions in this section and in Section 3 generalize for both monotonic and non-monotonic behavior.

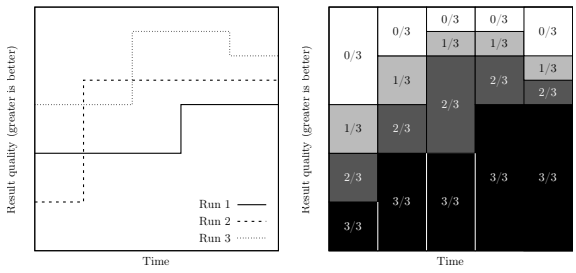

Figure 1: On the left the performance traces of three runs of an algorithm. On the right the corresponding performance profile for these runs.

Running the algorithm for different instances and/or executing it multiple times on the same instance, will likely result in different performance traces. Thus, to characterize the behavior of an algorithm over different runs we define a *performance profile*.

Definition 2.2 (Performance Profile). The *performance profile* of an algorithm a over a set of runs $R \subseteq \Omega_a$ is defined by a function

$$P_{a,R}(t, q) = \frac{1}{|R|} \sum_{r \in R} I\{K_{a,r}(t) \geq q\}$$

that denotes the probability of a run of the algorithm finding a solution of quality greater or equal to $q \in Q$ at time $t \in T$.

In Figure 1, on the left-hand side, we illustrate the performance traces of three runs of an algorithm, and on the right-hand side we show the corresponding performance profile over those runs. This is similar to a plot of the empirical attainment function [14].

The definition of a performance profile as a conditional probability function has been previously considered in [9], denoted by *run time distribution*, and in [2] through the use of the *empirical attainment function* [8]. However, these definitions consider the probability of finding a solution of quality greater or equal to q at, or before (whereas ours is only at) execution time t. Note that our definition of a performance profile is equivalent to these definitions if one assumes monotonic behavior.

Lastly, we note that the aggregation of different performance traces into a single profile should be carefully planned [9, 21]. For example, if the considered instances have different solution quality domains, then their aggregation may not be meaningful. One possibility to mitigate this issue is, for example, to consider the relative quality of a solution with respect to the optimal.

3 MEASURING THE PERFORMANCE OF AN ANYTIME ALGORITHM

The definition of a performance measure, which describes the performance of an anytime algorithm as a single value, is a key aspect for the development of an automatic selection methodology. Moreover, we are also interested in defining such a performance measure with respect to the anytime preferences of the decision maker. In this work, we assume that the anytime preferences of a decision maker are characterized by a utility function $w : T \times Q \rightarrow \mathbb{R}_0^+$ that for every combination of time and solution quality returns a non-negative scalar value that describes its preferences.

In the following, we start by describing logical relations between performance profiles. Then, we formulate performance measures for particular domains of T and Q, and for a bounded function w. Lastly, we discuss the properties of these measures with respect to the aforementioned logical relations. To make the reading easier we will denote a performance profile $P_{a,R}$ simply by P.

3.1 Logical Relations

The following logical relation between two performance profiles with respect to a utility function w is considered:

Definition 3.1 ($\geq_w$ pre-order). Given two performance profiles P and P', and a utility function $w : T \times Q \rightarrow \mathbb{R}_0^+$, we define the relation $P \geq_w P'$ iff for every $t \in T$ and $q \in Q$ where $w(t, q) > 0$, it holds that $P(t, q) \geq P'(t, q)$.

The above relation induces a partial order. Thus, we can define the following relations between two performance profiles:

$$P =_w P' \iff P \geq_w P' \wedge P' \geq_w P \quad \text{(Equality)}$$

$$P >_w P' \iff P \geq_w P' \wedge P' \ngeq_w P \quad \text{(Superiority)}$$

$$P \parallel_w P' \iff P \ngeq_w P' \wedge P' \ngeq_w P \quad \text{(Incomparability)}$$

The usefulness of these relations is linked to the characterization of the performance profiles. In particular, for an algorithm selection methodology, we are interested in comparing algorithms to solve a particular instance. Then, we should compare performance profiles that describe the expected behavior of an algorithm on that instance.

3.2 Performance Measures

In this section we are interested in defining performance measures that characterize a performance profile as a single (scalar) value. These measures should take into account the anytime preferences of the decision maker. Moreover, we argue that it is desirable that such performance measures are order preserving with respect to the $\geq_w$ pre-order described in the previous section, and that we are able to differentiate between incomparable profiles, i.e. allow incomparable profiles to have different performance measure values, as it is likely that the anytime algorithms considered for selection have incomparable performance profiles.

Up until this point, we have considered general domains T and Q, as well as a generally defined utility function w. However, to define our performance measures we need to restrict their definitions. In particular, we assume that domains T and Q are either continuous or discrete. An example of a continuous domain T is CPU-time, while an example of a discrete domain T is the number of function evaluations. To guarantee that our measures return a finite value we restrict the definition of utility function w to a region bounded by finite lower and upper bounds on time and quality, as follows

$$w(t, q) \rightarrow \begin{cases} \mathbb{R}_0^+ & \text{if } (t_\ell \leq t \leq t_u) \wedge (q_\ell \leq q \leq q_u) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where t_ℓ and t_u are the lower and upper bounds on time, and q_ℓ and q_u are the lower and upper bounds on quality.

We show in Table 1, the performance measures for the different combinations of continuous and discrete domains, as well as, values of t_ℓ, t_u, q_ℓ, and q_u, with respect to a helper function

$$h_{w,P}(t, q) = w(t, q)P(t, q) \quad (2)$$

where P is the performance profile whose performance we wish to measure, and w is a utility function describing the anytime preferences of the decision maker according to Equation 1.

The measure for continuous domains T and Q, and strict inequalities between the lower and upper bounds, is similar to the notion of weighted hypervolume discussed in [15] to measure the quality of a performance trace. The main differences are that our definition allows for algorithms with non-monotonic behavior, and that it is defined for a performance profile rather than a performance trace.

To characterize and study these measures we focus on the case where both domains T and Q are discrete. For this case, the performance measure defines a weighted sum over all the values of $P(t, q)$ within the bounded region of w. As such, when comparing two performance profiles with this measure we not only consider how often the value of $P(t, q)$ is better for one profile than for the other, but also how much better, or worse, the value is at each point. In the following, we show that the measure for discrete domains T and Q is order preserving with respect to the $\geq_w$ pre-order.

Proposition 3.2. *Given the performance measure*

$$M_w(P) = \sum_{t \in T} \sum_{q \in Q} h_{w,P}(t, q)$$

it holds that M_w is order preserving, formally

$$P \geq_w P' \implies M_w(P) \geq M_w(P')$$

Proof. If $P \geq_w P'$, then, from Definition 3.1, it holds that

$$w(t, q) > 0 \implies h_{w,P}(t, q) \geq h_{w,P'}(t, q)$$
$$w(t, q) = 0 \implies h_{w,P}(t, q) = h_{w,P'}(t, q)$$

Therefore, $h_{w,P}(t, q) \geq h_{w,P'}(t, q)$ for all $t \in T$ and $q \in Q$, which implies that $M_{w,P} \geq M_{w,P'}$. □

Similarly, it can be shown that M_w is strictly order preserving

$$P >_w P' \implies M_w(P) > M_w(P')$$
$$P =_w P' \implies M_w(P) = M_w(P')$$

We also show that the same measure can provide different values for incomparable algorithms.

Proposition 3.3. *Given the performance measure*

$$M_w(P) = \sum_{t \in T} \sum_{q \in Q} h_{w,P}(t, q)$$

and two performance profiles P and P', then it holds that

$$P \parallel_w P' \not\implies M_w(P) = M_w(P')$$

Table 1: Performance measures for the combinations of discrete and continuous domains for time and quality, as well as bounds t_ℓ, t_u, q_ℓ, q_u for utility function w. The measures are defined with respect to a helper function $h_{w,P}$ defined in Equation 2.

	Continuous T, Continuous Q	Continuous T, Discrete Q	Discrete T, Continuous Q	Discrete T, Discrete Q
$(t_\ell < t_u) \wedge (q_\ell < q_u)$	$\int_Q \int_T h_{w,P}(t,q)\ dt\ dq$	$\sum_{q\in Q} \int_T h_{w,P}(t,q)\ dt$	$\sum_{t\in T} \int_Q h_{w,P}(t,q)\ dq$	$\sum_{t\in T} \sum_{q\in Q} h_{w,P}(t,q)$
$(t_\ell = t_u) \wedge (q_\ell < q_u)$	$\int_Q h_{w,P}(t_\ell,q)\ dq$	$\sum_{q\in Q} h_{w,P}(t_\ell,q)$	$\int_Q h_{w,P}(t_\ell,q)\ dq$	$\sum_{q\in Q} h_{w,P}(t_\ell,q)$
$(t_\ell < t_u) \wedge (q_\ell = q_u)$	$\int_T h_{w,P}(t,q_\ell)\ dt$	$\int_T h_{w,P}(t,q_\ell)\ dt$	$\sum_{t\in T} h_{w,P}(t,q_\ell)$	$\sum_{t\in T} h_{w,P}(t,q_\ell)$
$(t_\ell = t_u) \wedge (q_\ell = q_u)$	$h_{w,P}(t_\ell,q_\ell)$	$h_{w,P}(t_\ell,q_\ell)$	$h_{w,P}(t_\ell,q_\ell)$	$h_{w,P}(t_\ell,q_\ell)$

Proof. Let $T = \{0,1\}$, $Q = \{0\}$, and $w(t,q) = 1$ for all $t \in T$ and $q \in Q$. Then, consider values $P(0,0) = 3, P(1,0) = 3$ for a performance profile P, and $P'(0,0) = 1, P'(1,0) = 4$ for a performance profile P'. For these values it holds that $P \parallel_w P'$ and $M_w(P) \neq M_w(P')$. □

The propositions above similarly generalize for the remaining performance measures in Table 1.

4 ALGORITHM SELECTION

The algorithm selection problem [18] concerns the selection of an algorithm to solve an instance, such that a performance measure, set by the decision maker, is optimized. We recall, from Section 1, two relevant aspects to consider for the selection with respect to anytime algorithms are: (i) to define a performance measure for anytime algorithms that takes the anytime preferences of the decision maker into account; and (ii) to include the anytime preferences of the decision maker into the selection problem at the time of selection. In the previous section we focused on the first issue. For this section, we turn into the second. First, we formalize the algorithm selection problem in the context of anytime algorithms. Then, we discuss possible approaches to develop a selection methodology that considers the anytime preferences at the time of selection.

4.1 Algorithm Selection Problem for Anytime Algorithms

In this work we formally describe a selection methodology by a mapping $S(\pi, w) \rightarrow A$, that for a given instance π and utility function w returns an algorithm $a \in A$. We remark that utility function w can change between calls to the selection task. The performance of a selection mapping is considered with respect to the performance of the selected algorithm a on the same instance and utility function. Thus, an "optimal" selection methodology can be defined as follows

$$S^*(\pi, w) = \operatorname*{argmax}_{a\in A} M_w(P_{a,\pi})$$

where $P_{a,\pi}$ denotes the performance profile of algorithm $a \in A$ on instance π, and $M_w(P_{a,\pi})$ is the performance measure of that profile with respect to the utility function w.

Devising an optimal selection methodology is often infeasible since there is usually no algorithm that, for all instances and utility functions, will always show the best performance. Moreover, precisely characterizing the behavior of the algorithms is often impossible since many factors can affect their performance. As such, we are interested in designing methodologies that can solve the algorithm selection problem approximately, and that will often return an algorithm with optimal, or close to optimal, performance.

4.2 Algorithm Selection Approaches

Different approximate selection methodologies can be found in the literature [10, 11]. However, up to our knowledge, none consider the dynamic anytime preferences of the decision maker. Nevertheless, the methodologies found in the literature can generally be divided into two categories: (i) regression approaches, which predict the (scalar) performance measure of the algorithms for an instance and then use these predictions to make the selection, e.g. [12, 20]; and (ii) classification approaches, which return the selected algorithm without predicting the performance of the algorithms, e.g. [17, 19]. In this section, we consider how the dynamic anytime preferences impact the feasibility of similar methodologies. Note that selection approaches often rely on *instance features* to guide the selection, which are features computed from the instance that will expectedly impact the performance of the algorithms, e.g. the number of decision variables for optimization problems. The study of instance features is, in itself, a challenging task which is still subject to active research for various optimization problems [3, 10].

The introduction of dynamic anytime preferences into the selection problem introduces some challenges for the design of a selection methodology. Assume a scenario with static anytime preferences, i.e. the preferences do not change between calls to the selection methodology. A reasonable regression approach is to learn to predict the performance measure for the algorithms from a set of training instances, based on instance features. To consider dynamic anytime preferences in a similar approach, we could introduce features related to the utility function w, e.g. the values of the bounds t_ℓ, t_u, q_ℓ, q_u, and features related to the output values of the utility function. However, this introduces several issues. First, the use of more features typically increases the complexity of machine learning techniques. Second, it may lead to a significant increase in the number of training examples needed as we have to consider a diverse set of utility functions. Assuming n training instances and m training utility functions, there are $n \cdot m$ possible training examples to consider. Last, creating a diverse set of m training utility functions is likely not trivial, since there is likely a very large, possibly

infinite, number of utility functions that the decision maker can give to the selection methodology.

A classification system can potentially limit the number of utility functions needed to learn the system, since it does not need to predict the performance of the algorithms. For example, consider the comparison of two algorithms: an heuristic that quickly achieves a good solution but stops on a local optimum, versus an exact algorithm that slowly starts with worse solutions but eventually reaches the global optimum. Then, for each instance, consider a time point t for which the exact algorithm is always better than the heuristic. If the classification system can learn to predict point t, then it no longer needs to be trained for utility functions where $t < t_\ell$ since it knows that the exact algorithm can always be selected in that case. Similar ideas for the other bounds can further help to reduce the number of utility functions needed to train the classification system. However, the number of remaining utility functions after these cuts may still be unfeasible. Moreover, these cuts are not always easy to identify, e.g. when considering many different algorithms.

To avoid the need of any utility function during the training phase, we consider a third approach. In particular, given an instance π and a utility function w, the selection methodology learns to predict a performance profile for each algorithm with respect to instance π. Then, at the time of selection, the methodology predicts the performance profile for each algorithm, and calculates the performance measure with respect to the utility function w for each predicted performance profile. Finally, it selects the algorithm whose performance profile has the best performance measure.

In Section 5.3, we present a methodology based on this approach that uses the instance features of instance π in order to identify similar instances in the training set. Then, the available performance traces for the identified instances are used to predict the performance profile for each algorithm with respect to instance π.

5 EXPERIMENTAL STUDY

In this section we present an experimental study on the selection between an exact and a heuristic algorithm for a bi-objective knapsack problem. The purpose of this preliminary study is to formalize, and study, a concrete selection methodology based on the last approach discussed in Section 4.2. We start by introducing the problem, instances, and the two algorithms considered. Afterwards, we formalize the selection methodology, and report the results of this methodology on various selection scenarios.

5.1 Benchmark Problem

We consider the bi-objective binary knapsack problem (BOBKP), defined by

$$\max \ (f_1(x), f_2(x)) = \left(\sum_{j=1}^{n} x_j v_j^1, \ \sum_{j=1}^{n} x_j v_j^2 \right) \tag{3}$$

$$\text{s.t.} \ \sum_{j=1}^{n} x_j c_j \leq C \tag{4}$$

$$x \in \{0,1\}^n \tag{5}$$

where n is the number of items, $v_j^1, v_j^2, c_j \in \mathbb{Z}^+$ denote the values and cost associated with an item $j = 1, \ldots, n$, respectively, $C \in \mathbb{Z}^+$ denotes a cost constraint, and x is the decision vector that describes which items are selected or not, in particular $x_j = 1$ denotes that the jth item is selected and $x_j = 0$ otherwise.

The instances considered for this study follow the generation procedure described in [1], which considers a cost constraint

$$C = \frac{1}{2} \sum_{j=1}^{n} c_j$$

and four instance types:

- Type A (Random instances): values v_j^1 and v_j^2, and costs c_j are randomly generated with respect to the uniform distribution in the range $[1, 1000]$;
- Type B (Unconflicting instances): values v_j^1 are randomly generated with respect to the uniform distribution in the range $[111, 1000]$, values v_j^2 in the range $[v_j^1 - 100, v_j^1 + 100]$, and the costs c_j in the range $[1, 1000]$;
- Type C (Conflicting instances): values v_j^1 are randomly generated with respect to the uniform distribution in the range $[1, 1000]$, values v_j^2 in the range $[\max\{900 - v_j^1, 1\}, \min\{1100 - v_j^1, 1000\}]$, and the costs c_j in the range $[1, 1000]$;
- Type D (Conflicting instances with correlated weight): same as Type C, except that the costs c_j are randomly generated in the range $[v_j^1 + v_j^2 - 200, v_j^1 + v_j^2 + 200]$.

A solution to a bi-objective problem consists of a set of mutually non-dominated decision vectors. Under the notion of Pareto optimality [6] this means that no decision vector in the solution set is better or equal than another in all objective values. The optimal solution, namely the Pareto set, is the set of all feasible decision vectors that are not dominated by any other feasible decision vector. To measure the quality of a solution set as a scalar value, different quality indicators have been proposed [23]. We note that the choice of a quality indicator is up to the decision maker, and that the selection methodology proposed in Section 5.3 does not depend on any specific indicator. However, it assumes that the values returned by the chosen indicator follow a total order. Still, the choice of an indicator should be carefully considered since it impacts the anytime data collected, which in turn influences which algorithm should be selected. In this work, we consider the hypervolume indicator [22] which corresponds to the area dominated by the image of a solution in the objective space with respect to a reference point. We consider a reference point $(-1, -1)$ so that any solution to the BOBKP has a non-zero quality value. The hypervolume was, in part, chosen due to being monotonic with respect to the notion of Pareto optimality [23]. As such, a better solution, according to the notion of Pareto dominance, will have a greater hypervolume value. Moreover, the hypervolume value is maximal for the Pareto set.

5.2 Algorithms

In this study we consider two algorithms: an exact algorithm, and a heuristic algorithm. This should provide a natural scenario for algorithm selection in an anytime context since we expect the heuristic algorithm to quickly find good solutions but to naturally stop before finding the Pareto set, whereas we expect the exact algorithm to initially find solutions with worst quality than those found by the heuristic, but to eventually surpass the heuristic and

find the Pareto set. We note that these assumptions may not always hold, e.g. the exact approach may take too long to be useful or a heuristic may not naturally stop. Nonetheless, they hold for the algorithms we consider in this study. In particular, we consider the "B-DP1" dynamic programming (DP) algorithm described in [7], and a Pareto local search (PLS) approach [16].

The DP approach is described in Algorithm 1. The methods ComputeNSDE and ComputeFromPrevious follow the procedures described in [1] and [7] as appropriate for variant "B-DP1". If interrupted the algorithm returns the non-dominated vectors from the latest set X_i that was computed. As such, to gather the anytime performance data required for the experimental study we measure the hypervolume of X_i at the end of each iteration.

Algorithm 1: Dynamic Programming

```
// Initialize C_0 with empty solution
X_0 ← {(0, ..., 0)}
F ← ComputeNSDE()
for i ∈ {1, 2, ..., n} do
    X_i ← ComputeFromPrevious(X_{i−1}, F)
return X_n
```

The PLS approach is described in Algorithm 2. The sets X_u and X_a, which correspond to the set of all unexplored non-dominated decision vectors and the set of all non-dominated decision vectors respectively, are implemented using a self-balancing binary tree data structure where the decision vectors are sorted according to the value of the first objective function f_1. For the Select method we consider a procedure that alternates between selecting the first and last decision vector in the unexplored set. We chose this procedure since it generally showed, in our preliminary tests, a better anytime behavior when compared to both a method that randomly selects a decision vector from the unexplored set, and a method that selects the decision vector with the maximal optimistic hypervolume improvement (OHI) heuristic described in [5]. The OHI corresponds to the "gap", in terms of hypervolume, around a decision vector in the objective space. Thus, a larger OHI value suggests a potentially greater increase in hypervolume by neighboring decision vectors. The neighborhood of a decision vector s is comprised of all feasible decision vectors within a hamming distance of 1 of the current decision vector (Neighborhood1), and all feasible decision vectors that result from the exchange of two values in the current decision vector (Neighborhood2). The neighborhood exploration follows the 1-flip-exchange method described in [13]. First, the algorithm explores the decision vectors of Neighborhood1. Then, if no new non-dominated decision vector was found during the exploration of Neighborhood1, the algorithm explores the decision vectors of Neighborhood2. The set of initial non-dominated decision vectors is comprised of the empty decision vector where $x_i = 0$ for all $i = 1, 2, \ldots, n$. It is worth noting that, for this neighborhood exploration definition, the PLS approach may not find the complete Pareto set, since it can fall into a Pareto local optimum set [16]. If interrupted the algorithm returns the set X_a. Thus, to measure its anytime performance we update the hypervolume of the set X_a in an online fashion whenever we remove or insert decision vectors.

Algorithm 2: Pareto Local Search

```
Input: X_0 (Set of initial non-dominated decision vectors)
X_u ← X_0
X_a ← X_0
while X_u ≠ ∅ do
    x ← Select(X_u)
    X_u ← X_u \ {x}
    aux ← 0
    for x' ∈ Neighborhood1(x) ∪ Neighborhood2(x) do
        if x' ∈ Neighborhood2(x) ∧ aux = 1 then
            break
        if IsDominated(x', X_a) then
            continue
        X_u ← RemoveDominated(X_u, x')
        X_a ← RemoveDominated(X_a, x')
        X_u ← X_u ∪ {x'}
        X_a ← X_a ∪ {x'}
        if x' ∈ Neighborhood1(x) then
            aux ← 1
return X_a
```

5.3 Selection Methodology

In the following, we present a selection methodology for the BOBKP based on the last approach proposed in Section 4. We recall that this approach consists of predicting the performance profile of a new instance for each algorithm, applying a performance measure to each predicted profile, and then selecting the algorithm whose profile presents the best performance measure value.

To predict the performance profile we take into account the performance traces of each algorithm on a set of training instances I. For an instance $i \in I$ we compute the following instance features:

- λ_1^i: the problem size, i.e. n in Equation 3;
- λ_2^i: the rank correlation coefficient between vectors v^1 and v^2;
- λ_3^i: the rank correlation coefficient between vectors v^1 and c;
- λ_4^i: the rank correlation coefficient between vectors v^2 and c.

To select the algorithm for an (unseen) instance π, we start by extracting the instance features λ^π. Then, to choose which training instances are used to build the predicted performance profile we consider the following procedure. First, we select the training instances that have the closest problem size to instance π, formally

$$I_s = \underset{i \in I}{\operatorname{argmin}} \left| \lambda_1^i - \lambda_1^\pi \right|$$

where $i \in I$ denotes an instance in the training set. Then, we compute the sum of the absolute differences between correlation coefficients, formally

$$s_i = \left| \lambda_2^i - \lambda_2^\pi \right| + \left| \lambda_3^i - \lambda_3^\pi \right| + \left| \lambda_4^i - \lambda_4^\pi \right|$$

for every training instance $i \in I_s$, and collect the m instances with smaller s_i into a set I_π.

From set I_π, which contains the training instances chosen for predicting the performance profile of instance π, we build a performance profile P_{a,I_π} with the available runs on those instances for

each algorithm $a \in A$. Then, the selection is done by applying the performance measure to the performance profile for each algorithm, and selecting the algorithm with the best performance, formally

$$S(\pi, w) = \operatorname*{argmax}_{a \in A} M_w \left(P_{a,I_\pi}\right) \tag{6}$$

where M_w denotes a performance measure with respect to a utility function w describing the decision maker's anytime preferences.

5.4 Experimental Scenario and Results

In this section we present the experimental scenario and the results of our proposed methodology with respect to the quality of the predicted performance measure and quality of the selection.

5.4.1 Experimental Scenario. For our experimental scenario we consider the four instance types discussed in Section 5.1 for the BOBKP, with problem sizes $n \in \{50, 60, \ldots, 140, 150\}$ for the training instances, and sizes $n \in \{50, 55, \ldots, 145, 150\}$ for the testing instances. For each combination of instance type and problem size, 30 random instances were generated for training and 10 for testing. We record the performance traces in terms of CPU time, in seconds, and relative hypervolume, which corresponds to the ratio between the current hypervolume and the maximal hypervolume. The maximal hypervolume corresponds to the hypervolume of the image of the Pareto set in the objective space, which is found by letting the DP algorithm finish its execution. For each instance, we record three runs of each algorithm to account for small fluctuations in CPU-time. Only three runs were considered since the algorithms are deterministic and the fluctuations are generally very small.

The DP and PLS algorithms shown in Section 5.2 were implemented in C++ and the experiments were carried out on a computer with an Intel i7-8700 CPU with clock frequency 3.20 GHz. Calculating the hypervolume throughout the execution of the algorithms introduces some overhead. In particular, for the instances considered in the experimental study, we saw an impact of less than 1% on the performance of the DP algorithm, and less than 5% for the PLS algorithm, which we consider to be within an acceptable margin.

For utility function w we consider the following definition

$$w(t, q) = \begin{cases} 1 & \text{if } (t_\ell \leq t \leq t_u) \wedge (q_\ell \leq q \leq q_u) \\ 0 & \text{otherwise} \end{cases}$$

The experiments are conducted with fixed values of $t_\ell = 0$, $q_\ell = 0$, $q_u = 1$, and varying values of $t_u \in [10^{-5}, 10^6]$ in order to study the methodology over varying anytime preferences. Note that solution quality is defined in the interval $[0, 1]$ since we consider a relative measure. As such, this definition of w indicates that the decision maker is equally likely to interrupt the algorithm for any time $t_\ell \leq t \leq t_u$ and for all possible quality values. For t_u we sample 200 points evenly spaced on a log scale of the interval $[10^{-5}, 10^6]$. Lastly, we consider the choice of the m-value for the presented selection methodology, which controls the number of training instances used to predict the performance profile of the selection instance. Our preliminary experiments suggest that $5 \leq m \leq 10$ provides the best results in terms of selection accuracy and quality of the predicted performance profile. As such, we arbitrarily consider $m = 7$ within that range for the experiments shown in this study.

To recap, for testing there are 21 different instance sizes, and 4 instance types. Then, for each combination of instance size and type, 10 instances are considered for testing. This results in a total of $21 \times 4 \times 10 = 840$ testing instances. Finally, for each instance we consider 200 possible anytime preferences by varying t_u. As such, there are a total of $840 \times 200 = 168\,000$ selection scenarios.

5.4.2 Prediction Quality. We discuss the prediction quality of our methodology with respect to a *relative performance measure* that denotes the ratio between the performance measure of a performance profile P and an "optimal" performance profile P^*, formally

$$\frac{M_w(P)}{M_w(P^*)}$$

where the "optimal" performance profile is defined by $P^*(t, q) = 1$ for every $t \in T$ and $q \in Q$. Note that, given the selection scenarios described in the previous section and the domains of quality and time, the performance measure is, from Table 1, given by

$$M_w(P) = \int_Q \int_T h_{w,P}(t, q)\, dt\, dq$$

Figure 2 shows the relative performance measure for the performance profile predicted by our proposed methodology, versus the "real" performance profile of the algorithm for a given instance. The results are reported for testing instances with problem size $n = 115$, a size that is not in the set of training instances. For a better visualization, we arbitrarily chose a single instance for each type. The figure suggests that the value of the relative performance measure for the predicted performance profile is often close to the same value for the real performance profile. However, we can observe a significant error for the PLS algorithm on instance type B. In Figure 3 we report the frequency of the error between the relative performance measure of the real and predicted profiles, over all selection scenarios. The plots shows that, although the error is very close to zero for most scenarios, there are still scenarios with a significant error. As such, we consider that there is still room to improve the predicted performance profile, for example by considering more instance features.

5.4.3 Selection Quality. We report in Table 2 the proportion of selection scenarios with an error in the specified intervals. The error is reported with respect to the difference in relative performance measure between the real performance profiles of the selected and optimal algorithms. An error with value zero (third column) indicates a correct selection and the proportion represents the accuracy of the methodology. Thus, a greater proportion is better. An error greater than zero (columns 4 to 6) indicates a wrong selection, thus a smaller proportion is better. The best proportions for each type and error interval are highlighted in bold. We report the results for: (i) our proposed methodology (Section 5.3); (ii) a random methodology that selects at random between the DP and PLS algorithms; (iii) a methodology that always selects the DP algorithm; and (iv) a methodology that always selects the PLS algorithm.

The results show that the accuracy of our proposed selection methodology is greater than 96% for the considered scenarios, and that it is always better than the remaining methodologies. The proportion of scenarios with an error within each specified interval is also lower for our proposed methodology. However, there are scenarios for instances of type B where our methodology shows an error greater than 10%, which we consider to be significant.

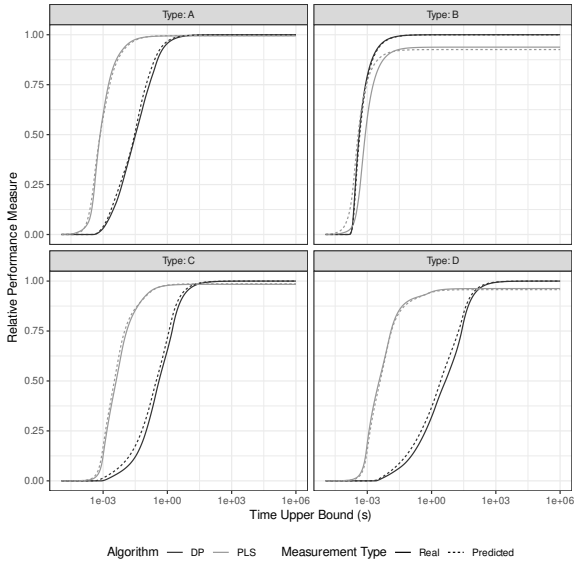

Figure 2: Relative performance measure for the "real" performance profiles (continuous lines) of testing instances of size $n = 115$, and for the performance profile (discontinuous lines) predicted by our selection methodology.

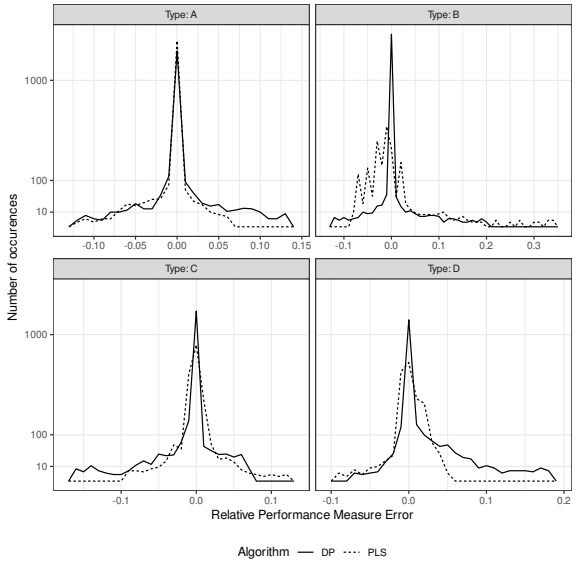

Figure 3: Absolute error between the relative performance measure of the predicted and real performance profiles.

6 CONCLUSION

In this work we discussed several aspects related to the algorithm selection problem for anytime algorithms. In particular, we presented performance measures that take into account the anytime preferences of the decision maker, and discussed possible approaches to develop a selection methodology which considers the decision maker's dynamic anytime preferences. Then, we carried out a preliminary study for the selection between an exact and a heuristic algorithm on a bi-objective knapsack problem, using a selection methodology developed following one of the discussed approaches.

The results of this preliminary study show that our selection methodology has an accuracy greater than 96% on the selected scenarios. However, we highlighted some scenarios where the error between the performance measure of the predicted and optimal algorithms was significant. We expect that this error can be reduced by having a better prediction of the performance profile for the selection instance. As such, a possible direction for future work is to extend this preliminary study by considering, for example, more instance features and different methods to select which training instances are used to predict the performance profile. Moreover, to better understand this selection methodology, different selection scenarios could be considered, e.g. scenarios with different utility functions and more algorithms. Lastly, another direction for future work is to study and apply algorithm selection methodologies of anytime algorithms on real-world scenarios.

Table 2: Proportion of selection scenarios with an error between the selected and optimal algorithms, in the presented intervals. Note that, an error of 0 (third column) indicates a correct selection. The best values are highlighted in bold.

Type	Methodology	[0]	(0, 0.01]	(0.01, 0.1]	(0.1, 1]
A	Proposed	**0.969**	**0.030**	**0.001**	**0.000**
	Random	0.498	0.290	0.076	0.136
	DP	0.479	0.122	0.126	0.272
	PLS	0.521	0.452	0.027	**0.000**
B	Proposed	**0.969**	**0.009**	**0.019**	**0.003**
	Random	0.501	0.028	0.368	0.104
	DP	0.814	0.043	0.076	0.067
	PLS	0.186	0.012	0.663	0.139
C	Proposed	**0.971**	**0.023**	**0.006**	**0.000**
	Random	0.502	0.154	0.193	0.150
	DP	0.453	0.141	0.105	0.300
	PLS	0.547	0.169	0.283	**0.000**
D	Proposed	**0.989**	**0.008**	**0.004**	**0.000**
	Random	0.503	0.067	0.222	0.208
	DP	0.378	0.121	0.085	0.416
	PLS	0.622	0.014	0.364	**0.000**

ACKNOWLEDGMENTS

This work was partially supported by PICS project MOCO-SEARCH co-funded by the French National Center for Scientific Research (CNRS) and the Portuguese Foundation for Science and Technology (FCT). The first author acknowledges the FCT for Ph.D. studentship SFRH/BD/132275/2017 co-funded by the European Social Fund and by the State Budget of the Portuguese Ministry of Education and Science. This work was partially funded by national funds through the FCT within the scope of the project CISUC - UID/CEC/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020.

REFERENCES

[1] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. 2009. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers & Operations Research* 36, 1 (Jan. 2009), 260–279. https://doi.org/10.1016/j.cor.2007.09.009

[2] Marco Chiarandini. 2005. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems.* Ph.D. Dissertation. Technical University of Darmstadt, Darmstadt, Germany.

[3] Fabio Daolio, Arnaud Liefooghe, Sébastien Verel, Hernán Aguirre, and Kiyoshi Tanaka. 2017. Problem Features versus Algorithm Performance on Rugged Multiobjective Combinatorial Fitness Landscapes. *Evolutionary Computation* 25, 4 (Winter 2017), 555–585. https://doi.org/10.1162/evco_a_00193

[4] Thomas L. Dean and Mark S. Boddy. 1988. An Analysis of Time-Dependent Planning. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence (AAAI'88).* AAAI Press, 49–54.

[5] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. 2015. Anytime Pareto local search. *European Journal of Operational Research* 243, 2 (June 2015), 369–385. https://doi.org/10.1016/j.ejor.2014.10.062

[6] Matthias Ehrgott. 2005. *Multicriteria Optimization* (2nd ed.). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-27659-9

[7] José Rui Figueira, Luís Paquete, Marco Simões, and Daniel Vanderpooten. 2013. Algorithmic improvements on dynamic programming for the bi-objective {0,1} knapsack problem. *Computational Optimization and Applications* 56, 1 (Sept. 2013), 97–111. https://doi.org/10.1007/s10589-013-9551-x

[8] Viviane Grunert da Fonseca, Carlos M. Fonseca, and Andreia O. Hall. 2001. Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function. In *Evolutionary Multi-Criterion Optimization (EMO 2001).* Springer, Berlin, Heidelberg, 213–225. https://doi.org/10.1007/3-540-44719-9_15

[9] Holger H. Hoos and Thomas Stützle. 2005. *Stochastic Local Search: Foundations & Applications.* Morgan Kaufmann, San Francisco, CA. https://doi.org/10.1016/B978-1-55860-872-6.X5016-1

[10] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation* 27, 1 (Spring 2019), 3–45. https://doi.org/10.1162/evco_a_00242

[11] Lars Kotthoff. 2016. Algorithm Selection for Combinatorial Search Problems: A Survey. In *Data Mining and Constraint Programming*. Springer, Cham, 149–190. https://doi.org/10.1007/978-3-319-50137-6_7

[12] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. 2003. A Portfolio Approach to Algorithm Selection. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03).* 1542–1543.

[13] Arnaud Liefooghe, Luís Paquete, Marco Simões, and José R. Figueira. 2011. Connectedness and Local Search for Bicriteria Knapsack Problems. In *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2011).* Springer, Berlin, Heidelberg, 48–59. https://doi.org/10.1007/978-3-642-20364-0_5

[14] Manuel López-Ibáñez, Luís Paquete, and Thomas Stützle. 2010. Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization. In *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin, Heidelberg, 209–222. https://doi.org/10.1007/978-3-642-02538-9_9

[15] Manuel López-Ibáñez and Thomas Stützle. 2014. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research* 235, 3 (June 2014), 569–582. https://doi.org/10.1016/j.ejor.2013.10.043

[16] Luis Paquete, Tommaso Schiavinotto, and Thomas Stützle. 2007. On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research* 156 (Aug. 2007), 83–97. https://doi.org/10.1007/s10479-007-0230-0

[17] Sergey Polyakovskiy, Mohammad Reza Bonyadi, Markus Wagner, Zbigniew Michalewicz, and Frank Neumann. 2014. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14).* Association for Computing Machinery, New York, NY, USA, 477–484. https://doi.org/10.1145/2576768.2598249

[18] John R. Rice. 1976. The Algorithm Selection Problem. In *Advances in Computers*. Vol. 15. Elsevier, 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

[19] Matheus Guedes Vilas Boas, Haroldo Gambini Santos, Luiz Henrique de Campos Merschmann, and Greet Vanden Berghe. 2019. Optimal decision trees for the algorithm selection problem: integer programming based approaches. *International Transactions in Operational Research* (Sept. 2019). https://doi.org/10.1111/itor.12724

[20] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research* 32 (June 2008), 565–606. https://doi.org/10.1613/jair.2490

[21] Shlomo Zilberstein. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17, 3 (Fall 1996), 73–83. https://doi.org/10.1609/aimag.v17i3.1232

[22] Eckart Zitzler. 1998. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* Ph.D. Dissertation. Swiss Federal Institute of Technology Zurich.

[23] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. 2003. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation* 7, 2 (April 2003), 117–132. https://doi.org/10.1109/TEVC.2003.810758